Listing of Claims:

1. (Original) A queuing system comprising:

a queue having a plurality of addressable storage locations associated therewith;

queue logic to control write operations to said queue, said queue logic operatively configured to write data events to said queue in a re-circulating sequential manner irrespective of whether previously stored data has been read out;

a current event counter updated by said queue logic to keep track of a count value that corresponds to the total number of data events written to said queue, said current event counter capable of counting an amount that is greater than said plurality of addressable storage locations;

read logic operatively configured to read data events from said queue according to a prescribed manner, said read logic further communicably coupled to said current event counter for reading said count value stored therein; and

a read pointer controlled by said read logic that relates to where in said queue data is to be read from said queue, wherein said read logic can read from said queue based upon said read pointer independently of write operations to said queue.

2. (Original) The queuing system according to claim 1, further comprising a previous event counter that is controlled by said read logic that relates to a previous value of said current event counter at the time of a previous read operation on said queue by said read logic.

3. (Original) The queuing system according to claim 2, wherein said read logic is further operatively configured to determine whether data has been lost in the queue due to queue overflow based upon a comparison of a current value of said current event counter and said previous value stored in said previous event counter.

4. (Original) The queuing system according to claim 2, wherein said read logic is further operatively configured to perform read operations according to a first prescribed manner when

no queue overflow is detected, and said read logic performs read operations from said queue according to a second prescribed manner when queue overflow is detected.

5. (Original) The queuing system according to claim 2, wherein said read pointer is derived directly from a predetermined number of the lowest order bits of said previous event counter.

6. (Original) The queuing system according to claim 1, wherein a write pointer is derived directly from a predetermined number of the lowest order bits of said current event counter.

7. (Currently Amended) The queuing system according to claim 1, wherein said queue logic ~~stores for each data event, a combination of~~ merges a sequence number derived from ~~the value of said current event counter value~~ said count value of said current event counter ~~and said data element in~~ with each data event stored in said queue.

8. (Original) The queuing system according to claim 1, wherein said user communicates with said queue through an intermediate interface.

9. (Original) The queuing system according to claim 1, wherein said read logic is associated with a plurality of users, each user comprising:

read logic operatively configured to read information from said queue according to a prescribed manner, said read logic further communicably coupled to said current event counter for reading said count value stored therein; and

a read pointer updated by said read logic that relates to where in said queue data is to be read from said queue, wherein said read logic can read from said queue based upon said read pointer independently of write operations to said queue, wherein said read pointer and read logic of each user operates independently of one another.

10. (Original) The queuing system according to claim 1, wherein said read logic further cascades data events read from said queue into a local queue for subsequent processing.

11. (Original) A queuing system comprising:

a queue having a predetermined number of addressable storage locations;

an event counter operatively configured to sequentially update a count value stored therein each time a data event is written into said queue, said count value capable of storing a maximum count that exceeds said predetermined number of addresses;

a write pointer that is derived from said count value stored in said event counter from which a select addressable storage location of said queue can be determined for temporarily storing each data event that is to be queued;

a read pointer from which a desired addressable storage location of said queue can be identified for a read operation;

queue logic communicably coupled to said queue, said event counter, and said write pointer to control writing new data events to said queue; and

read logic coupled to said queue, said event counter and said read pointer, said read logic operatively configured to read from said queue in a first manner when no overflow of said queue is detected, and to read from said queue in a second manner when overflow of said queue is detected.

12. (Original) The queuing system according to claim 11, wherein said write pointer is encoded into said event counter.

13. (Original) The queuing system according to claim 11, wherein said predetermined number of addresses of said queue is defined by the expression: $m = 2n$, where m is the number of addresses, and n is a positive integer.

14. (Original) The queuing system according to claim 13, wherein said write pointer is defined by the lowest n bits of said event counter.

15. (Original) The queuing system according to claim 14, wherein a portion of said count value is stored in said queue with each data event written thereto, said portion defined by at least one bit of said count value starting at the n+1 bit.

16. (Original) The queuing system according to claim 11, wherein each read from the queue is nondestructive, and each write to the queue overwrites the current content of the storage location addressed by the write pointer.

17. (Original) The queuing system according to claim 11, wherein said first manner of reading from said queue comprises reading from said queue in a first in, first out manner such that a list sequential read follows a temporal aging from the oldest to the newest events in said queue and said second manner of reading from said queue comprises reading from said queue in a manner that reads the most recent events written to the queue first.

18. (Original) The queuing system according to claim 11, wherein said second manner of reading from said queue comprises setting said read pointer to said write pointer prior to initiating a read operation.

19. (Original) The queuing system according to claim 11, wherein said second manner of reading from said queue comprises reading in a first direction for a predetermined portion of a read cycle, and reading in a second direction for a remainder portion of said read cycle.

20. (Original) The queuing system according to claim 11, wherein said second manner of reading from said queue comprises setting said read pointer to a position a predetermined number of addresses in a direction opposite to said write pointer, and beginning a read cycle wherein a plurality of data events are read in the direction of write operations to said queue.

21. (Original) The queuing system according to claim 11, wherein said read logic further comprises a previous event counter that keeps track of a representation of said count value of said event counter at the time of a previous read operation.

22. (Original) The queuing system according to claim 21, wherein said read counter is derived from the lowest order bits of said previous event counter.

23. (Original) A method of queuing data comprising:

defining a queue having addressable storage locations associated therewith;

keeping track of a current count value that corresponds to the total number of data events written to said queue, said current count value capable of counting an amount that is greater than the number of said addressable storage locations;

keeping track of a write pointer that corresponds to a position in said queue for a write operation thereto;

writing data events to said queue in a re-circulating sequential manner irrespective of whether previously stored data has been read out; and

for each user associated with said queue:

keeping track of a previous count value that corresponds to said current count value at the time of a previous access to said queue thereby; and

reading from said queue according to a prescribed manner.

24. (Original) The method according to claim 23, wherein at least one user reads from said queue according to a first prescribed manner when no queue overflow has been detected, and reads from said queue according to a second prescribed manner when overflow has been detected.

25. (Original) The method according to claim 24, wherein queue overflow is detected if the difference between said current count value and said previous count value is greater than a total number of said addressable storage locations.

26. (Original) The method according to claim 24, wherein a select user requests data events from said queue comprising:

specifying a total number of requested data events;

specifying a timeout period that represents the maximum amount of time said user is willing to wait for said data events

accessing said queue to obtain said data events; and

queuing any extracted data events in a local queue accessible by said user.

27. (Original) The method according to claim 24, wherein a select user reads from said queue comprising:

reading said current count value;

reading said previous count value;

comparing said current count value to said previous count value to determine whether overflow has occurred to the queue with respect to said user;

if no overflow is detected:

reading a queued data event based upon said read pointer;

updating said read pointer based upon said first predetermined manner; and

updating said previous event counter value; and

if overflow is detected:

updating said read pointer based upon a second predetermined manner;

reading at least one queued data event based upon said read pointer;

updating said read pointer based upon said second predetermined manner; and

updating said previous event counter value.

28. (Original) The method according to claim 27, further comprising after each read of said queue where no overflow is detected:

determining a new current count value;

comparing said new current count value to said previously stored count value to determine whether overflow has occurred; and if overflow is detected:

updating said read pointer based upon a second predetermined manner;

reading at least one queued data event based upon said read pointer;

updating said read pointer based upon said predetermined manner; and

updating said previous event counter value.

29. (Original) The method according to claim 27, wherein said read pointer is updated to a new position that corresponds to a current value of said write pointer.